Word Embeddings

(Pre-Transformers)

Dan Salo

October 27th, 2021

Fill In the Blank

1. “If I don’t leave now, I’ll be late for ____.”
2. “If I don’t leave now, I’ll be late for ____. Tacos are my favorite”
3. “If I don’t leave now, I’ll be late for ____. Tacos are my favorite midday meal.”

“If I don’t leave now, I’ll be late for **lunch**. Tacos are my favorite midday meal.”

1. “Here comes the ____”
2. “Here comes the ____, and I say, ‘It’s all right.’”

“Here comes the **sun**, and I say, ‘It’s all right’”

Fill In the Blank -- Reflections

- **Context is King!** The first blank was iteratively further constrained with more context.
- The **Distributional Hypothesis** (1954) states that words with similar meanings tend to occur in similar contexts. “You shall know a word by the company it keeps.”
- **Transfer learning in action**: bringing knowledge from a different domain (i.e. the Beatles) to improve performance on a specific task (i.e. fill in the blank).

https://www.tandfonline.com/doi/pdf/10.1080/00437956.1954.11659520

Motivation

Word Embeddings are vectorized representations of words.

What properties are desirable?

- Fast to produce
- Dense vectors encoded with semantic meaning
- Simple to use in downstream tasks

NLP 101

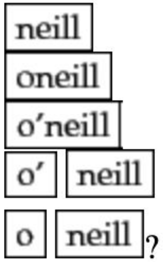
Tokenization 101

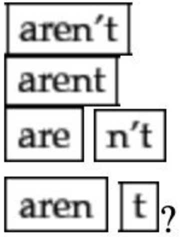
“Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.”

Q: How can we subdivide this text so its context can be compared to other contexts?

A: *Tokenization* is the process of splitting text into tokens, a useful semantic unit.

Basic tokenization is rules-based:





spaCy

NLTK

https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html

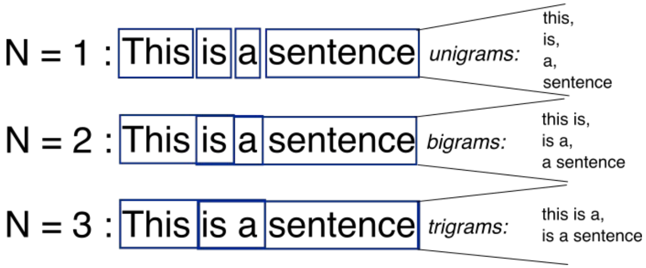
N-Grams 101

What’s a “gram”? A character, token, syllable, etc.

1. “Unigram”
2. “Bigram”
3. “Trigram”
4. “4-gram”

The power of N-Grams:

- Count up the occurrence of n-grams for an estimate of language probability
- “is a”



N-Grams “chunk” text

https://www.cbrinton.net/ECE20875-2020-Spring/W10/ngrams.pdf
https://lena-voita.github.io/nlp_course/language_modeling.html#markov_property

Language Modeling 101

Example: “If I don’t leave now, I’ll be late for _____.”
Q: What’s the relative probability of the blank being filled with class vs lunch?

p(class | *If I don’t leave now, I’ll be late for*) ≈
p(class | *for*) * p(for | *late*) * p(late | *be*) * … p(I | *If*)

$$p(w_1, \cdots, w_T) = \prod_i p(w_i \mid w_{i-1}, \cdots, w_{i-n+1})$$

p(lunch | *If I don’t leave now, I’ll be late for*) ≈
p(lunch | *for*) * p(for | *late*) * p(late | *be*) * … p(I | *If*)

p(class | …) / p(lunch | …) ≈ p(class | for) / p(lunch | for)

$p(w_i \mid w_{i-1})$ can be estimated via bigram counts for a large, relevant corpus.
p(class | for) / p(lunch | for) = **count(“for class”) / count(“for lunch”)**

https://www.cbrinton.net/ECE20875-2020-Spring/W10/ngrams.pdf
https://lena-voita.github.io/nlp_course/language_modeling.html#markov_property

sklearn.feature_exracation.text.CountVectorizer

Bag of Words Vectorization

- One-hot encoding: corpus vocabulary-length vector of all 0's except one [0,0,0,1,0,0,0,0,...0]
- Term-Doc matrix: Binary vector of documents in which the word appears

Characteristics

- Simple, intuitive, fast
- Local context is not preserved
- All terms weighted equally
- Sparse, vocabulary-length vectors

Example of text data: Titles of Some Technical Memos

c1: *Human* machine *interface* for ABC *computer* applications
c2: A *survey* of *user* opinion of *computer system response time*
c3: The *EPS user interface* management *system*
c4: *System* and *human system* engineering testing of *EPS*
c5: Relation of *user* perceived *response time* to error measurement

m1: The generation of random, binary, ordered *trees*
m2: The intersection *graph* of paths in *trees*
m3: *Graph minors* IV: Widths of *trees* and well-quasi-ordering
m4: *Graph minors*: A *survey*

	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

https://people.eng.unimelb.edu.au/mbouadjenek/papers/wordembed.pdf

Co-Occurrence Vectorization

The Counts matrix calculates the co-occurence of words within the context window, e.g. the document or sentence.

Characteristics

- Simple, intuitive, fast
- Local context is *partially* preserved
- All terms weighted equally
- Sparse, vocabulary-length vectors

1. I enjoy flying.
2. I like NLP.
3. I like deep learning.

$$
\begin{array}{c c}
 & \begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\
\begin{array}{c} I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array} &
\left[\begin{array}{cccccccc}
0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0
\end{array}\right]
\end{array}
$$

https://people.eng.unimelb.edu.au/mbouadjenek/papers/wordembed.pdf

Naive Vectorization

Drawbacks

- Large *and* sparse vectors
- Poor use of contextual information within corpus

Desired

- Compact *and* dense vectors
- Better use of contextual information with corpus

Word Embedding Evolution

- (1988) **Latent Semantic Analysis**: term-weight based model
- (2013) **Word2Vec**: prediction model
- (2014) **GLoVe**: counts model
- (2018) **ELMo**: language model-based

Latent Semantic Analysis

Deerwester, Scott, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman.
"Indexing by latent semantic analysis."
Journal of the American Society for Information Science 41, no. 6 (1990): 391-407.

sklearn.feature_exracation.text.TfidfVectorizer

TF-IDF

Characteristics

- A re-weighting of the term-doc matrix
- As term frequency increases and

Strength(s)

- Upweights unique terms in a document
- Common usage in open-source search engines (Elasticsearch)

Drawback(s)

- Contextual nature of vectorization dilutes as document length increases
- Weightings are corpus-dependent
- Sparse, vocabulary-length vectors

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF
Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

The Equation

$\mathtt{tf}(t,d)$

	blue	bright	can	see	shining	sky	sun	today
1	1/2	0	0	0	0	1/2	0	0
2	0	1/3	0	0	0	0	1/3	1/3
3	0	1/3	0	0	0	1/3	1/3	0
4	0	1/6	1/6	1/6	1/6	0	1/3	0

x

$\mathtt{idf}(t,D)$

	blue	bright	can	see	shining	sky	sun	today
	0.602	0.125	0.602	0.602	0.602	0.301	0.125	0.602

$$\mathtt{tfidf}(t,d,D) = \mathtt{tf}(t,d) \cdot \mathtt{idf}(t,D)$$

	blue	bright	can	see	shining	sky	sun	today
1	**0.301**	0	0	0	0	0.151	0	0
2	0	0.0417	0	0	0	0	0.0417	**0.201**
3	0	0.0417	0	0	0	**0.100**	0.0417	0
4	0	0.0209	**0.100**	**0.100**	**0.100**	0	0.0417	0

An Example Vectorization

https://ted-mei.medium.com/demystify-tf-idf-in-indexing-and-ranking-5c3ae88c3fa0
https://sci2lab.github.io/ml_tutorial/tfidf/

Latent Semantic Analysis

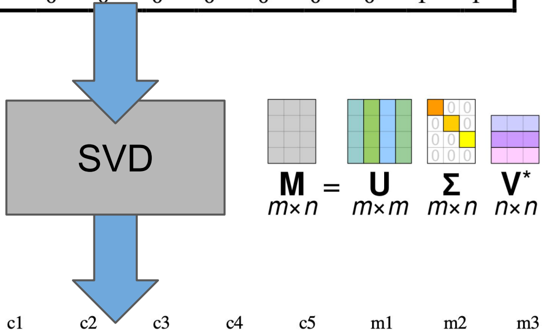
Characteristics

- Applies SVD to the term-doc matrix (or tf-idf matrix)
- *U* are doc latent factors, *V* are term latent factors (i.e. word vectors)
- Word vectors “borrow” information from other documents in which word is not present

Clarifications

- SVD assumes a Gaussian distribution of terms
- Factors aren’t interpretable
- Adding new documents / terms requires recomputation

	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1



	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	0.16	0.40	0.38	0.47	0.18	-0.05	-0.12	-0.16	-0.09
interface	0.14	0.37	0.33	0.40	0.16	-0.03	-0.07	-0.10	-0.04
computer	0.15	0.51	0.36	0.41	0.24	0.02	0.06	0.09	0.12
user	0.26	0.84	0.61	0.70	0.39	0.03	0.08	0.12	0.19
system	0.45	1.23	1.05	1.27	0.56	-0.07	-0.15	-0.21	-0.05
response	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
time	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
EPS	0.22	0.55	0.51	0.63	0.24	-0.07	-0.14	-0.20	-0.11
survey	0.10	0.53	0.23	0.21	0.27	0.14	0.31	0.44	0.42
trees	-0.06	0.23	-0.14	-0.27	0.14	0.24	0.55	0.77	0.66
graph	-0.06	0.34	-0.15	-0.30	0.20	0.31	0.69	0.98	0.85
minors	-0.04	0.25	-0.10	-0.21	0.15	0.22	0.50	0.71	0.62

Rank 2 approximation of term-doc matrix ($U_T \Sigma_T V_T^*$)

http://lsa.colorado.edu/papers/dp1.LSAintro.pdf

LSA Embeddings

Achievements over Naive

- Not long or sparse vectors ✅
- Better use of contextual information within corpus ✅

Drawbacks

- SVD is prohibitively expensive with large matrices
  though optimizations are available (See Brian's Netflix talk)

Desired

- Faster method for contextual, dense word vectors

https://nlp.stanford.edu/IR-book/html/htmledition/latent-semantic-indexing-1.html

Word2Vec

T Mikolov, I Sutskever, K Chen, GS Corrado, J Dean. Distributed representations of words and phrases and their compositionality. NIPS 2013.
https://arxiv.org/pdf/1301.3781.pdf
https://arxiv.org/pdf/1310.4546.pdf

Neural Language Models

A Neural Probabilistic Language Model, Bengio *et al*, 2003

The approach pursues language modeling and produces word vectors as a by-product:

1. associate with each word in the vocabulary a distributed *word feature vector* (a real-valued vector in $\mathbb{R}^m$),
2. express the joint *probability function* of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the *word feature vectors* and the parameters of that *probability function*.

input/feature #1	input/feature #2	output/label
Thou	shalt	______

What's the probability of the next word being "not"?

https://jalammar.github.io/illustrated-word2vec/
https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

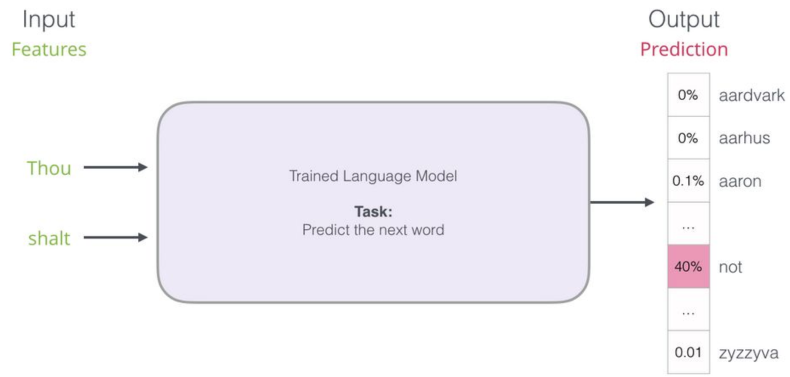
Neural Language Models

input/feature #1 input/feature #2 output/label

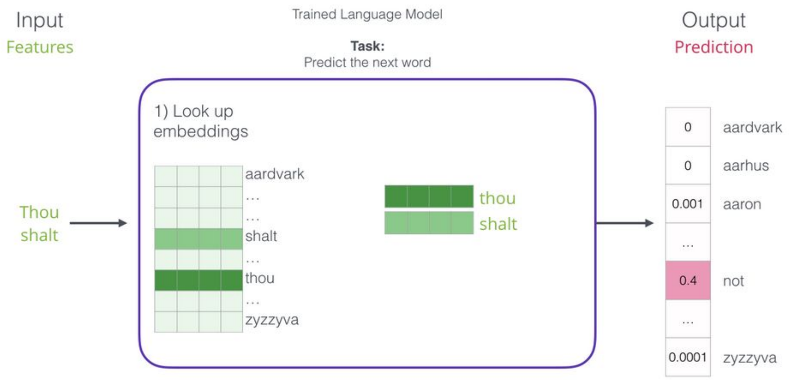
Thou shalt ______

LM Task: Predict Next Word
i.e. Fill in the blank

$$p(w_1, \cdots, w_T) = \prod_i p(w_i \mid w_{i-1}, \cdots, w_{i-n+1})$$



LM Model and Prediction



Word Embeddings!

Word2Vec

Improves upon Bengio's 2003 model by:

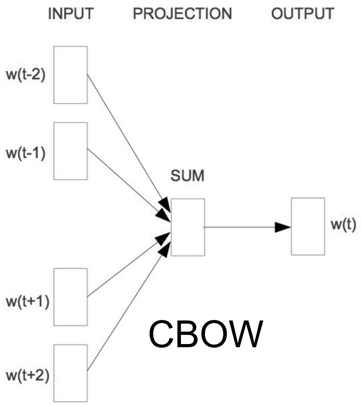
- Removing hidden neural network layer and non-linearities (tanh)
- Introduces *hierarchical sampling* and *negative sampling* for approximation of softmax at Output

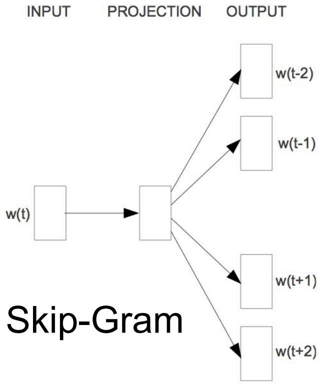
CBOW model:

- The distributed representations of context are combined to predict the word in the middle
- Produces vectors cluster by syntax (e.g. "cat" and "cats")

Skip-gram model:

- The distributed representation of the input word is used to predict the context
- Produces vectors clustered by semantics (e.g. "cat" and "dog")





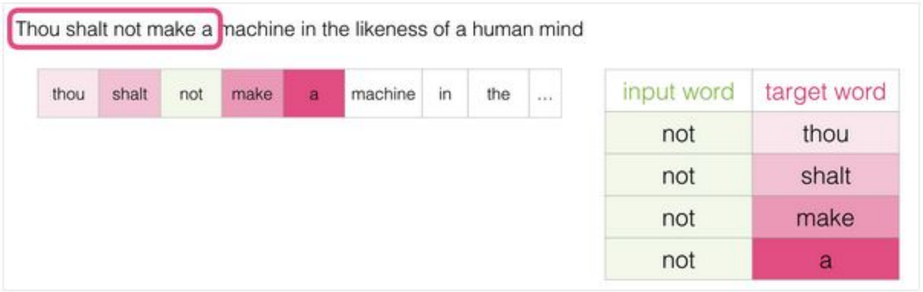
Skip-Gram Basics

Data

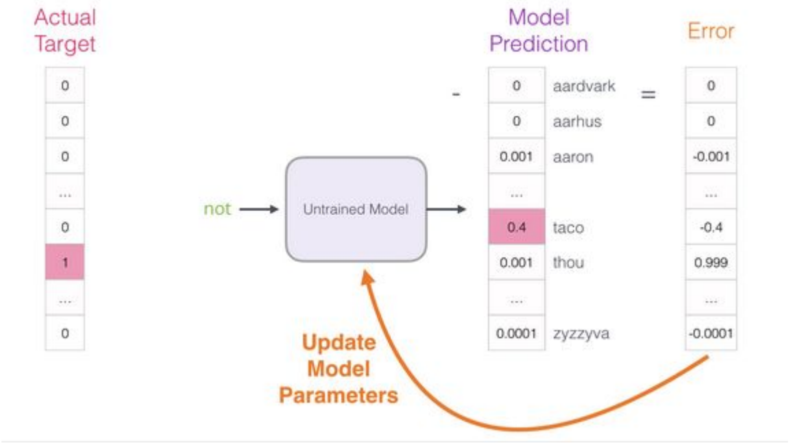
- The training samples are easily extracted from the corpus.
- Sliding window default is 5

Training

- Calculate softmax over sampled vocabulary
- Backprop the error to update the model params (i.e. word vectors)



Data



Training

Word2Vec Embeddings

Achievements over LSA

- Faster algorithms ✅
- Denser vectors ✅

Drawbacks

- Only includes local, windowed context during training and misses out on global occurrence statistics

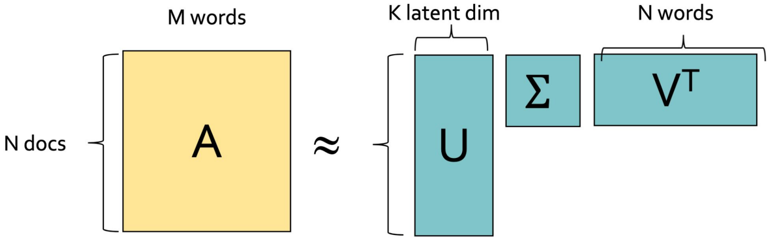
Desired

- Incorporation of global occurrence information as well

GLoVe

J Pennington, R Socher, C Manning.
GLoVe: Global Vectors for Word Representation.
ENMLP 2014.
https://nlp.stanford.edu/pubs/glove.pdf

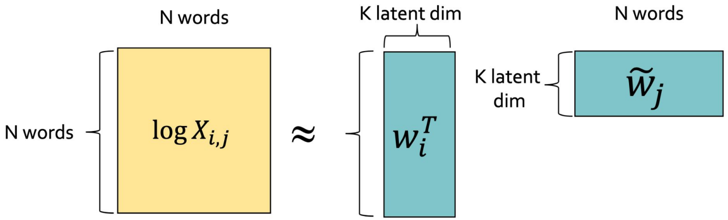
GLoVe



LSA: SVD on Term-Doc Matrix

Characteristics

- GLoVe = GLobal Vectors
- Applies Matrix Factorization to the co-occurrence matrix
- Generates two sets of word vectors differing by initialization
  (Authors average them to produce final vector set)



GLoVe: MF on Counts Matrix

Intuition

- The dot product of two word vectors should be proportional to their co-occurrence count.
- Dot product of orthogonal vectors = 0
  The vectors of words that do not co-occur should be orthogonal

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(\boxed{w_i^T \tilde{w}_j} + b_i + \tilde{b}_j - \boxed{\log X_{ij}}\right)^2$$

https://nlp.stanford.edu/pubs/glove.pdf

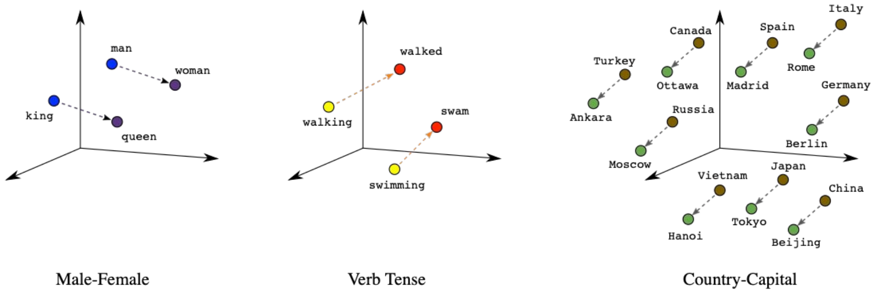
GLoVe vs Word2Vec vs LSA

LSA:

- Vector space does not support analogies
- Weighs all co-occurrences equally
- Better semantics on small datasets[1]

GLoVe and Skip-Gram Word2Vec:

- Similar performance on analogies
- Are interchangeable in many tasks



Quantifiable Analogies

Nearest Neighbors On Different Historical Corpa

https://nlp.stanford.edu/projects/histwords

[1]https://arxiv.org/abs/1610.01520

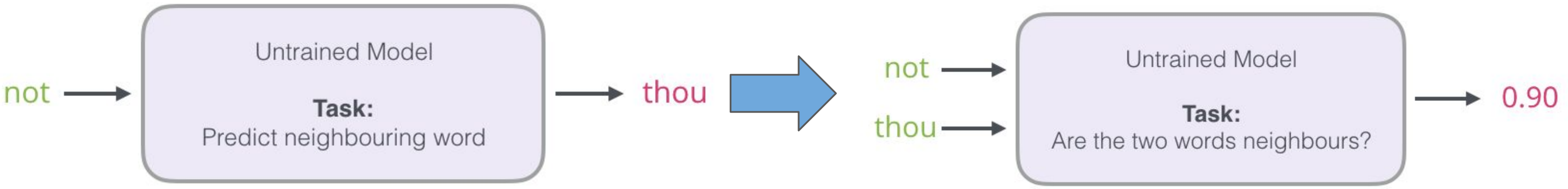
Others

Sense2Vec: incorporate POS and NER information into word vectors (“bat - NOUN”)

Doc2Vec: Word2Vec applied to documents instead of words

fastText: sets out to learn same LM Task but with key differences from word2vec

- Learns vectors for character *n*-grams and sums to produce word vectors
- Reframes expensive softmax as a binary classification problem. Faster!



https://arxiv.org/abs/1405.4053
https://arxiv.org/abs/1511.06388
https://arxiv.org/abs/1607.04606

GLoVe/Word2Vec Embeddings

Achievements

- Faster algorithms ✅
- Denser vectors ✅
- Use global/local contextual information ✅

Drawbacks

- Word vectors become overloaded with its various senses

Desired

- Disambiguate word sense on runtime context!

*I can't **trust** you.*

*They have no **trust** left for their friend.*

*He has a **trust** fund.*

ELMo

M Peters, *et al*.
Deep contextualized word representations
NAACL 2018.
https://arxiv.org/abs/1802.05365

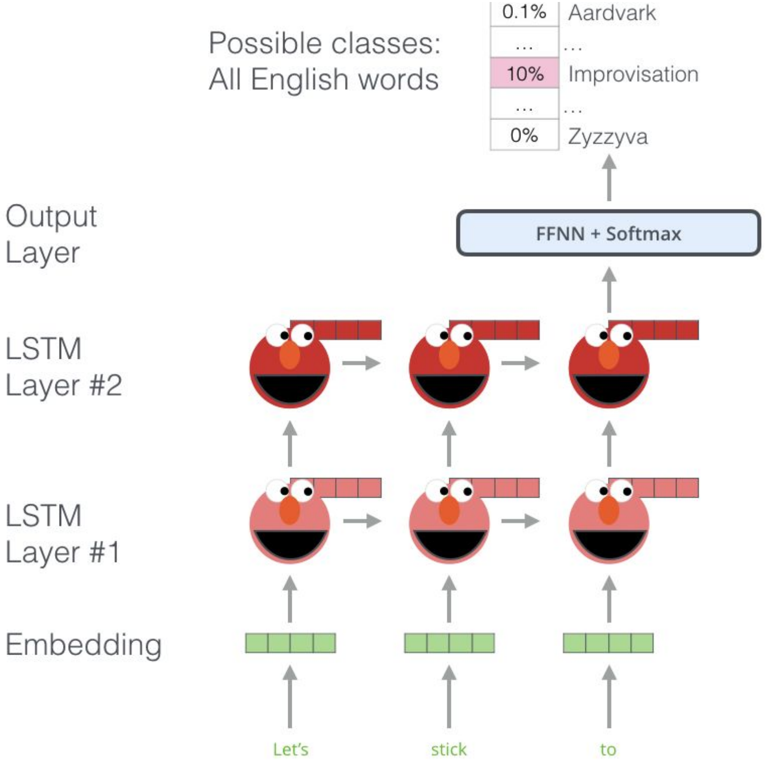
Recurrent Neural Networks for Language Modeling

input/feature #1 input/feature #2 output/label

Thou shalt ______

LM Task: Predict Next Word
i.e. Fill in the blank

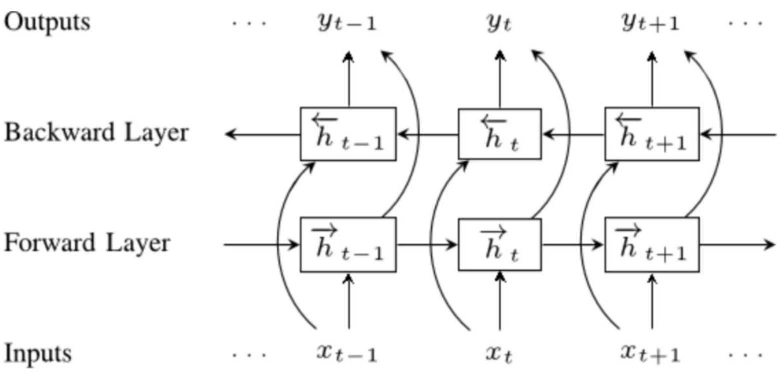
$$p(w_1, \cdots, w_T) = \prod_i p(w_i \mid w_{i-1}, \cdots, w_{i-n+1})$$

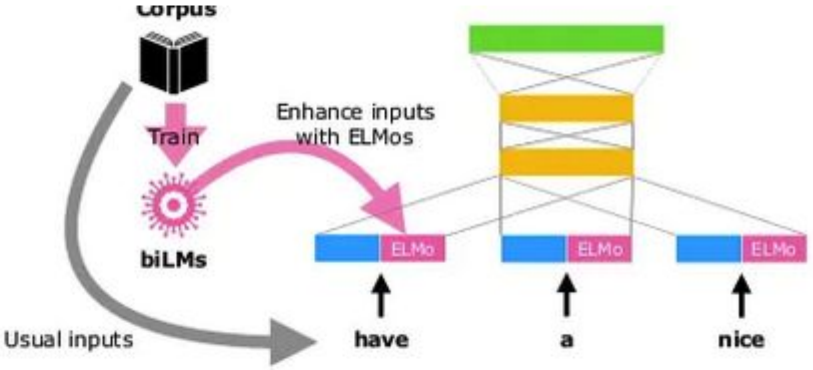


https://jalammar.github.io/illustrated-bert/

Bidirectional LSTM

- Views forward *and* backward context
- Also referred to as biLM in the paper (bidirectional Language Model)





Training and Knowledge Incorporation

https://paperswithcode.com/method/elmo

ELMo Takeaways

1. Encode corpus information in a *model* rather than in dense *vectors*
2. Use *model* to compute *vectors* at runtime
3. Incorporate vectors into downstream model

But…

What if we could use the information-laden model *as the model itself for the task?* Eliminate the downstream model and fine-tune (like an ImageNet model).

Highlighted References

Word2Vec / fastText:

- Pretty pictures: https://jalammar.github.io/illustrated-word2vec/
- A little math: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
- More math: https://ruder.io/word-embeddings-1/index.html

Word Embeddings:

- **Overview**: https://rbouadjenek.github.io/papers/wordembed_v2.0.pdf